Title

stata.com

window menu — Create menus

Syntax Description Remarks and examples Also see

Syntax

Clear previously defined menu additions

```
window menu clear
```

Define submenus

```
window menu append submenu "defined_menuname" "appending_menuname"
```

Define menu item

```
window menu append item "defined_menuname" "entry_text" "command_to_execute"
```

Define separator bars

```
window menu append separator "defined_menuname"
```

Activate menu changes

```
window menu refresh
```

Add files to the Open Recent menu

```
window menu add_recentfiles "filename" [ , rlevel(#) ]
```

The quotation marks above are required.

"*defined_menuname*" is the name of a previously defined menu or one of the user-accessible menus `"stUser"`, `"stUserData"`, `"stUserGraphics"`, or `"stUserStatistics"`.

Description

`window menu` allows you to add new menu hierarchies.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Overview

A menu is a list of choices. Each choice may be another menu (known as a submenu) or an item. When you click on an item, something happens, such as a dialog box appearing or a command being executed. Menus may also contain separators, which are horizontal bars that help divide the menu into groups of related choices.

Stata provides the top-level menus **Data**, **Graphics**, **Statistics**, and **User** to which you may attach your own submenus, items, or separators.

A menu hierarchy is the collection of menus and how they relate.

`window menu` allows you to create menu hierarchies, set the text that appears in each menu, set the actions associated with each menu item, and add separators to menus.

New menu hierarchies are defined from the top down, not from the bottom up. Here is how you create a new menu hierarchy:

1. You append to some existing Stata menu a new submenu using `window menu append submenu`. That the new submenu is empty is of no consequence.
2. You append to the new submenu other submenus, items, or separators, all done with `window menu append`. In this way, you fill in the new submenu you already appended in step 1.
3. If, in step 2, you appended submenus to the menu you defined in step 1, you append to each of them so that they are fully defined. This requires even more `window menu append` commands.
4. You keep going like this until the full hierarchy is defined. Then you tell Stata's menu manager that you are done using `window menu refresh`.

Everything you do up to step 4 is merely definitional. At step 4, what you have done takes effect.

You can add menus to Stata. Then you can add more menus. Later, you can add even more menus. What you cannot do, however, is ever delete a little bit of what you have added. You can add some menus and `window menu refresh`, then add some more and `window menu refresh`, but you cannot go back and remove part of what you added earlier. What you can do is remove all the menus you have added, restoring Stata to its original configuration. `window menu clear` does this.

So, in our opening example, how did the **Regression** submenu ever get defined? By typing

```
. window menu append submenu "stUserStatistics" "Regression"
. window menu append item "Regression" "Simple" ...
. window menu append item "Regression" "Multiple" ...
. window menu append item "Regression" "Multivariate" ...
. window menu refresh
```

`stUserStatistics` is the special name for Stata's **User–Statistics** built-in menu. The first command appended a submenu to `stUserStatistics` named **Regression**. At this point, **Regression** is an empty submenu.

The next three commands filled in **Regression** by appending to it. All three are items, meaning that when chosen, they invoke some Stata command or program. (We have not shown you what the Stata commands are; we just put "..." to indicate them.)

Finally, `window menu refresh` told Stata we were done and to make our new additions available.

Clear previously defined menu additions

`window menu clear`

clears any additions that have been made to Stata's menu system.

Define submenus

`window menu append submenu "`*defined_menuname*`" "`*appending_menuname*`"`

defines a submenu. This command creates a submenu with the text *appending_menuname* (the double-quote characters do not appear in the submenu when displayed) attached to the "*defined_menuname*". It also declares that the "*appending_menuname*" can later have further submenus, items, and separators appended to it. Submenus may be appended to Stata's built-in **User** menu using the command

```
window menu append submenu "stUser" "appending_menuname"
```

For example,

```
window menu append submenu "stUser" "New Menu"
```

appends **New Menu** to Stata's **User** menu. Likewise, submenus may be appended to the built-in submenus of **User**—**Data**, **Graphics**, and **Statistics**—by using **stUserData**, **stUserGraphics**, or **stUserStatistics** as the *defined_menuname*.

Define menu items

`window menu append item "`*defined_menuname*`" "`*entry_text*`" "`*command_to_execute*`"`

defines menu items. This command creates a menu item with the text "*entry_text*", which is attached to the "*defined_menuname*". When the item is selected by the user, "*command_to_execute*" is invoked.

For example,

```
window menu append item "New Menu" "Describe" "describe"
```

appends the menu item **Describe** to the **New Menu** submenu defined previously and specifies that if the user selects **Describe**, the `describe` command is to be executed.

Define separator bars

```
window menu append separator "defined_menuname"
```

defines a separator bar. The separator bar will appear in the position in which it is declared and is attached to an existing submenu.

For example,

```
window menu append separator "New Menu"
```

adds a separator bar to **New Menu**.

Activate menu changes

```
window menu refresh
```

activates the changes made to Stata's menu system.

Add files to the Open Recent menu

The **Open Recent** menu is a list of datasets recently used or saved by the user. Selecting a dataset from the menu causes Stata to execute a `use` command on the dataset to load the data. The datasets are represented in the list as the absolute path or URL to the dataset.

A dataset is added to the list if the dataset is loaded by the command `use` or saved by the command `save`. The list is ordered from the most recently used datasets to the least recently used datasets. The maximum number of datasets in the list is twenty and datasets are removed from the bottom of the list when the maximum is reached. If a dataset already exists in the list when it is to be added, the existing entry is moved to the top of the list.

The list of datasets from the **Open Recent** menu is saved when exiting Stata and loaded when starting Stata. Stata removes datasets that do not exist from the list when it exits and starts but not during a session. Stata does not attempt to determine if a URL is valid.

```
window menu add_recentfiles "filename" [ , rlevel(#) ]
```

adds a dataset to the **Open Recent** menu under the **File** menu. Only datasets should be added to the **Open Recent** menu.

To prevent temporary files from being added to the **Open Recent** menu, Stata does not add datasets used or saved by do-files and ado-files or when running a batch file. However, for the cases where you do wish to add a dataset used or saved by an ado-file or do-file, you may use the `rlevel()` option.

The `rlevel()` option determines the maximum run level an ado-file issuing the `window menu add_recentfiles` may run at for a dataset to be added to the **Open Recent** menu. If no ado-file is running, the run level is `0`. If an ado-file executes another ado-file which executes another ado-file before returning to the previous ado-file, the run level is `3`. `rlevel(0)` adds a dataset only if no ado-file or do-file is running and is the default. `rlevel(3)` adds a dataset if an ado-file is up to 3 levels deep when called. `rlevel(-1)` adds a dataset regardless of the run level and is the only way to add a dataset from a do-file.

For example, `sysuse` is implemented as an ado-file. We want to add datasets loaded by `sysuse` to the **Open Recent** menu only if the user entered `sysuse` from the command line. We add to `sysuse.ado`

```
window menu add_recentfiles "filename", rlevel(1)
```

If we had used a run level of 2, any dataset loaded by `sysuse` from an ado-file would be added to the **Open Recent** menu which is not what we want.

Keyboard shortcuts (Windows only)

When you define a menu item, you may assign a keyboard shortcut. A shortcut (or keyboard accelerator) is a key that allows a menu item to be selected via the keyboard in addition to the usual point-and-click method.

By placing an ampersand (`&`) immediately preceding a character, you define that character to be the shortcut. The ampersand will not appear in the menu item, but the character following the ampersand will be underlined to alert the user of the shortcut. The user may then choose the menu item by either clicking with the mouse or holding down *Alt* and pressing the shortcut key. Actually, you only have to hold down *Alt* for the top-level menu. For the submenus, once they are pulled down, holding down *Alt* is optional.

If you need to include an ampersand as part of the "*entry_text*", place two ampersands in a row.

It is your responsibility to avoid creating conflicting keyboard shortcuts. When the user types in a keyboard shortcut, Stata finds the first item with the defined shortcut.

Example:

```
window menu append submenu "stUserStatistics" "&Regression"
```

defines a new submenu named **Regression** that will appear in the **User**–**Statistics** menu and that users may access by pressing *Alt-U* (to open the **User** menu), then *S* (to open the **Statistics** menu), and finally *R*, the shortcut defined for **Regression**.

Examples

Below we use the `window menu` commands to add to Stata's existing top-level menu. The following may be typed interactively:

```
window menu clear
window menu append submenu "stUser" "&My Data"
window menu append item "My Data" "&Describe data" "describe"
window menu refresh
```

`window menu clear`
Clears any user-defined menu items and restores the menu system to the default.

`window menu append submenu "stUser" "&My Data"`
Appends to the **User** a new submenu called `My Data`. Note that you may name this new menu anything you like. You can capitalize its name or not. You may include spaces in it. The new menu appears as the last item on the **User** menu.

`window menu append item "My Data" "&Describe data" "describe"`
Defines a menu item (including a keyboard shortcut) named **Describe data** to appear within the **My Data** submenu. This name is what the user will actually see. It also specifies the command to execute when the user selects the menu item. In this case, we will run the `describe` command.

`window menu refresh`
Causes all the menu items that have been defined and appended to the default system menus to become active and to be displayed.

Advanced features: Dialogs and built-in actions

Recall that menu items can have associated actions:

`window menu append item "`*defined_menuname*`" "`*entry_text*`" "`*command_to_execute*`"`

Actions other than Stata commands and programs can be added to menus. In the course of designing a menu system, you may include menu items that will invoke dialogs, open a Stata dataset, save a Stata graph, or perform some other common Stata menu command.

You can specify "*command_to_execute*" as one of the following:

"DB *dialog_to_invoke*"
: invokes the dialog box defined by the file *dialog_to_invoke*`.dlg`. For example, specifying "`DB regress`" as the "*command_to_execute*" results in the dialog box for Stata's `regress` command being invoked when the item is selected.

"XEQ about"
: displays Stata's *About* dialog box. The About dialog box is accessible from the default system menu by selecting **About** from the **File** menu.

"XEQ save"
: displays Stata's *File Save* dialog box to save the dataset in memory. This dialog box is accessed from the default system menu by selecting **Save** from the **File** menu.

"XEQ saveas"
: displays Stata's *File Save As* dialog box to save the dataset in memory. This dialog box is accessible from the default system menu by selecting **Save As...** from the **File** menu.

"XEQ savegr"
: displays the *Save Stata Graph File* dialog box, which saves the currently displayed graph. This dialog box is accessible from the default system by selecting **Save Graph** from the **File** menu.

"XEQ printgr"
: prints the graph displayed in the Graph window. This is available in the default menu system by selecting **Print Graph** from the **File** menu. Also see [P] **window manage**.

"XEQ use"
: displays Stata's *File Open* dialog box, which loads a Stata dataset. This is available in the default menu system by selecting **Open...** from the **File** menu.

"XEQ exit"
: exits Stata. This is available from the default menu system by selecting **Exit** from the **File** menu (or selecting **Quit** from the **Stata** menu on Mac).

"XEQ conhelp"
: opens the Stata help system to the default welcome topic. This is available by clicking on the **Help!** button in the help system.

Advanced features: Creating checked menu items

command_to_execute in

`window menu append item "`*defined_menuname*`" "`*entry_text*`" "`*command_to_execute*`"`

may also be specified as `CHECK` *macroname*.

Another detail that menu designers may want is the ability to create checked menu items. A checked menu item is one that appears in the menu system as either checked (includes a small check mark to the right) or not.

"CHECK *macroname*" specifies that the global macro *macroname* should contain the value as to whether or not the item is checked. If the global macro is not defined at the time that the menu item is created, Stata defines the macro to contain zero, and the item is not checked. If the user selects the menu item to toggle the status of the item, Stata will place a check mark next to the item on the menu system and redefine the global macro to contain one. In this way, you may write programs that access information that you gather via the menu system.

Note that you should treat the contents of the global macro associated with the checked menu item as "read only". Changing the contents of the macro will not be reflected in the menu system.

Putting it all together

In the following example, we create a larger menu system. Note that each submenu defined using `window menu append submenu` contains other submenus and/or items defined with `window menu append item` that invoke commands.

```
——————————————————————————————————————————— begin lgmenu.do ———
capture program drop mylgmenu
program mylgmenu
   version 13
   win m clear
   win m append submenu "stUserStatistics" "&Regression"
   win m append submenu "stUserStatistics" "&Tests"
   win m append item "Regression" "&OLS" "DB regress"
   win m append item "Regression" "Multi&variate" "choose multivariate"
   win m append item "stUserGraphics" "&Scatterplot" "choose scatterplot"
   win m append item "stUserGraphics" "&Histogram" "myprog1"
   win m append item "stUserGraphics" "Scatterplot &Matrix" "choose matrix"
   win m append item "stUserGraphics" "&Pie chart" "choose pie"
   win m append submenu "Tests" "Test of &mean"
   win m append item "Tests" "Test of &variance" "choose variance"
   win m append item "Test of mean" "&Unequal variances" "CHECK DB_uv"
   win m append separator "Test of mean"
   win m append item "Test of mean" "t-test &by variable" "choose by"
   win m append item "Test of mean" "t-test two &variables" "choose 2var"
   win m refresh
end
capture program drop choose
program choose
   version 13
   if "'1'" == "by" | "'1'" == "2var" {
           display as result "'1'" as text " from the menu system"
           if $DB_uv {
                   display as text "  use unequal variances"
           }
           else {
                   display as text "  use equal variances"
           }
   }
   else {
                display as result "'1'" as text " from the menu system"
   }
end
capture program drop myprog1
program myprog1
   version 13
   display as result "myprog1" as text " from the menu system"
end
——————————————————————————————————————————— end lgmenu.do ———
```

Running this do-file will define a program `mylgmenu` that we may use to set the menus. Note that, other than the **OLS** item, which launches the `regress` dialog box, the menu items will not run any interesting commands, as the focus of the example is in the design of the menu interface only. To see the results, type `mylgmenu` in the Command window after you run the do-file. Below is an explanation of the example.

The command

```
win m append submenu "stUserStatistics" "&Regression"
```

adds a submenu named **Regression** to the built-in menu **Statistics** under the **User** menu. If the user clicks on **Regression**, we will display another menu with items defined by

```
win m append item "Regression" "&OLS" "DB regress"
win m append item "Regression" "Multi&variate" "choose multivariate"
```

Because none of these entries open further menus, they use the `item` version instead of the `submenu` version of the `window menu append` command.

Similarly, the built-in **User**–**Graphics** menu is populated using `window menu item` commands.

```
win m append item "stUserGraphics" "&Scatterplot" "choose scatterplot"
win m append item "stUserGraphics" "&Histogram" "myprog1"
win m append item "stUserGraphics" "Scatterplot &Matrix" "choose matrix"
win m append item "stUserGraphics" "&Pie chart" "choose pie"
```

For the **Tests** submenu, we decided to have one of the entries be another submenu for illustration. First, we declared the **Tests** menu to be a submenu of **User**–**Statistics** using

```
win m append submenu "stUserStatistics" "&Tests"
```

We then defined the entries that were to appear below the **Tests** menu. There are two entries: one of them is another submenu, and the other is an item. For the submenu, we then defined the entries that are below it.

Finally, note how the commands that are run when the user makes a selection from the menu system are defined. For most cases, we simply call the same program and pass an argument that identifies the menu item that was selected. Each menu item may call a different program if you prefer. Also note how the global macro that was associated with the checked menu item is accessed in the programs that are run. When the item is checked, the global macro will contain 1. Otherwise, it contains zero. Our program merely has to check the contents of the global macro to see if the item is checked or not.

Also see

[P] **dialog programming** — Dialog programming

[P] **window manage** — Manage window characteristics

[P] **window programming** — Programming menus and windows